# Shell

# 一. shell概述

在Linux内核中与用户之间的解释器程序;通常指/bin/bash;负责向内核翻译及传达用户/程序指令;相当于操作系统的"外壳".

## 1. shell脚本

shell脚本的关键在于输入多个命令并处理每个命令的结果,shell可以将多个命令串起来,一次执行.

## 2. 脚本构成

#! 脚本声明(使用哪种解释器)

# 注释信息(步骤,思路,用途,变量含义等)

可执行的语句

## 3. 使用方式

交互式(命令行):

人工干预,智能化程度高,逐条解释执行,效率低

非交互式(脚本):

需要提前设计,智能化难度大,批量执行,效率高

# 二. 变量

## 1. 定义

以固定名称存放可能会变化的值

## 2. 作用

允许临时性地将信息存储在shell脚本中;

提高脚本对任务的需求

## 3. 定义,查看

定义:

var=变量值

查看:

引用变量值: $变量名

查看变量值: echo $变量名

显示所有环境变量: env

显示所有变量: set

取消变量:

unset 变量名

变量初始值:

${ var:-值 }

## 4. 变量种类

系统变量:

**环境变量**

用来设置特定的系统信息.

**位置变量**

存储执行脚本时提供的参数.

|  |  |
| --- | --- |
| 变量名 | 含义 |
| $0 | 当前所在的进程或脚本 |
| $n | 在执行脚本时提供的参数 |

**预定义变量**

特殊用途的变量

|  |  |
| --- | --- |
| 变量名 | 含义 |
| $$ | 当前运行进程的PID号 |
| $? | 命令执行后的返回值 |
| $# | 已加载的位置变量的个数 |
| $\* | 所有位置变量的值 |

自定义变量:

由用户自主设置,修改及使用

## 5. 作用范围

全局变量:

在当前shell及子shell环境中均有效.

局部变量:

定义新的变量默认只在当前shell环境中有效.

e.g.

export 局部变量名=变量值 为局部变量添加全局属性

export -n 全局变量名 取消指定变量的全局属性

## 6. 定界符

双引号 " " :允许扩展,以$引用其他变量

单引号 ' ' :禁止扩展,即便$也视为普通字符

反撇号 ` ` :将命令的执行输出作为变量值

# 三. 数值运算

## 1. expr命令

功能: 允许在命令行上处理数学表达式,能够识别少数的数学和字符串操作.

|  |  |
| --- | --- |
| 操作符 | 描述 |
| ARG1 + ARG2 | 返回ARG1和ARG2的算术运算和 |
| ARG1 - ARG2 | 返回ARG1和ARG2的算术运算差 |
| ARG1 \* ARG2 | 返回ARG1和ARG2的算术乘积 |
| ARG1 / ARG2 | 返回ARG1被ARG2除算术商 |
| ARG1 % ARG2 | 返回ARG1被ARG2除算术余商 |

## 2. 使用方括号

在将一个数字运算结果赋值给某个变量时,可以使用美元符和方括号($[ operation ])将数学表达式围起来.

格式: $[ operation ]

说明:

-- 乘法操作\*无需转义,运算符两侧可以无空格

-- 引用变量可以省略$符号

e.g. 计算结果替换表达式本身,可结合echo命令输出

In a=43

echo $[a+21]

out 64

## 3. bc

功能: 允许在命令行中输入浮点表达式

e.g. scale内置变量

更改scale变量默认值;可保留小数点相应的位数

## 4. let命令

功能: 运算时不显示结果,可以方便的对变量进行自增减.

e.g. 默认自增1

let a++

# 四. 条件测试

## 1. 数值比较

|  |  |
| --- | --- |
| 比较 | 描述 |
| n1 -eq n2 | 检查n1是否与n2相等 |
| n1 -ge n2 | 检查n1是否大于或等于n2 |
| n1 -gt n2 | 检查n1是否大于n2 |
| n1 -lt n2 | 检查n1是否小于n2 |
| n1 -le n2 | 检查n1是否小于或等于n2 |
| n1 -en n2 | 检查n1是否与n2不等 |

## 2. 字符串比较

|  |  |
| --- | --- |
| 比较 | 描述 |
| str1 = str2 | 检查str1是否和str2相同 |
| str1 != str2 | 检查str1是否和str2不同 |
| str1 > str2 | 检查str1是否大于str2 |
| -z str | 检查str是否为空 |
| -n str | 检查str是否非空 |

## 3. 文件比较

|  |  |
| --- | --- |
| 比较 | 描述 |
| -e file | 检查file是否存在 |
| -d file | 检查file是否存在并且是一个目录 |
| -f file | 检查file是否存在并且是一个文件 |
| -s file | 检查file是否存在并且非空 |
| -r,-w,-x file | 检查file是否存在并且可读,可写,可执行 |
| -O file | 检查file是否存在并且属于当前用户所有 |
| -G file | 检查file是否存在并且默认组与当前用户相同 |
| fiel-1 -nt file-2 | 检查file-1是否比file-2新 |
| file-1 -ot file-2 | 检查file-1是否比file-2旧 |

## 4. 逻辑分隔符

|  |  |
| --- | --- |
| 操作符 | 描述 |
| test-1 && test-2 | 给定条件必须都成立,整个测试结果才能为真 |
| test-1 || test-2 | 满足其中一个条件,则整个测试结果为真 |

e.g. 当/opt/testdir目录不存在时,创建该目录

[ -d /opt/testdir ] && mkdir /opt/testdir

# 五. 选择结构语句

## 1. if-then

格式:

if command

then

commands

fi

说明:

bash shell的if语句会运行if后面的那个命令;

如果该命令的退出状态码是0,位于then部分的命令就会被执行;

如果该命令的退出状态码是其他值,then部分的命令就不会被执行;

bash shell会继续执行脚本中的下一个命令;

fi语句用来表示if-then语句到此结束.

e.g. 复合条件测试

[ condition1 ] && [ condition2 ]

[ condition1 ] || [ condition2 ]

## 2. if-then-else

格式:

if command; then

commands

else

commands

fi

说明:

如果该命令的退出状态码是0,位于then部分的命令就会被执行;

当if语句中的命令返回非零退出状态码时,bash shell会执行else部分中的命令.

## 3. if-then-elif

格式:

if command; then

commands

elif command; then

commands

else

commands

fi

## 4. case命令

格式:

case var in

pattern1) commands1;;

pattern2 | pattern3) commands;;

\*) default commands;;

esac

说明:

case命令会将指定的变量与不同模式进行比较;

如果变量和模式是匹配的,那么shell会执行该模式指定的命令;

通过竖线操作符在一行中分隔出多个模式;

星号会捕获所有与已知模式不匹配的值.

e.g. 编写方便使用nginx服务的工具脚本

#! /bin/bash

file\_path='/usr/local/nginx/sbin/nginx'

case $1 in

start)

natstat -ntulp | grep -q nginx

[ $? -eq 0 ] && echo "服务已经开启" &&　exit

$file\_path;;

stop)

$file\_path -s stop;;

re)

$file\_path -s stop

$file\_path;;

status)

netstat -ntulp | grep nginx

[ $? -eq 0 ] && echo "服务开启" || echo "服务未开启";;

\*)

echo "start | stop | re | status"

esac

命令: netstat -ntulp | grep nginx

功能: 查询网络数据结构

选项:

-n 以数字格式显示端口号

-t 显示TCP连接的端口

-u 显示UDP连接的端口

-l 显示服务正在监听的端口信息，如httpd启动后，会一直监听80端口

-p 显示监听端口的服务名称是什么（也就是程序名称）

# 六. 循环

## 1. for循环

格式:

for var in list

do

commands

done

说明:

在list参数中,需要提供迭代中要用到的一系列值;

在每次迭代中,变量var会包含列表中的当前值;

直到迭代到list中最后一个值,循环结束.

e.g. C语言风格的for循环

for ((a=1; a<10; a++))

do

printf "The next number is : $a"

done

## 2. while循环

格式:

while test command

do

other commands

done

说明:

while命令的关键在于所指定的then command的退出状态码必须随着循环中运行的命令而改变;

如果退出状态码不发生变化,while循环就会将一直不停地进行下去.

## 3. until循环

格式:

until test command

do

other commands

done

说明:

until命令和while命令工作的方式完全相反;

until命令要求你指定一个通常返回非零退出状态码的测试命令;

一旦测试命令返回了退出状态码0,循环就结束了.

# 七. 函数及中断控制

## 1. 含义:

函数是一个脚本代码块,可以为其命名并在代码中任何位置重用.

## 2. 定义与调用

### 定义:

**格式1**

function name {

commands

}

说明:

关键子function,后跟分配给改代码块的函数名;

name属性定义了赋予函数的唯一名称;

bash shell会按命令在函数中出现的顺序依次执行.

**格式2**

name( ) {

commands

}

### 调用:

像其他shell命令一样;在行中指定函数名.

说明:

函数中处理变量

全局变量

在shell脚本中任何地方都有效的变量.

局部变量

函数内部使用的任何变量都可以被声明成局部变量.

变量声明前加上local关键字

## 3. 中断及退出

break

跳出当前所在的循环体,执行循环体后的语句块.

continue

跳过循环体余下的语句,重新判断条件决定是否需要执行下一次循环.

exit

退出脚本,默认的返回值是0.

# 八. 字符串处理

## 1. 子串截取

${ str : begin : 长度 }

说明:

字符串索引下表从0开始,默认为0.

## 2. 子串替换

${ str/old/new } # 只替换第1个匹配结果

${ str//old/new } # 替换全部匹配结果

## 3. 按条件删除

从左向右:

${ str#\*关键字 } # 删除到最近匹配

${ str##\*关键字 } # 删除到最远匹配

说明:

str为字符串

\* 通配符

从右向左:

${ str%关键字\* } # 删除到最近匹配

${ str%%关键字\* } # 删除到最远匹配

# 九. 正则表达式

使用"一串字符"来描述有共同属性的数据.

## 1. egrep工具

格式:

egrep [选项] '正则表达式' 文件

前置命令 | egrep [选项] '正则表达式'

功能:

以行为单位,逐行进行处理;默认只输出与表达式相匹配的文本行.

选项:

-i 忽略字母大小写

-v 条件取反

-c 统计匹配的行数

-q 无任何输出,一般用来检测

-n 显示出匹配结果所在的行号

--color 标红显示匹配子串

基本正则符号列表

|  |  |
| --- | --- |
| **正则符号** | **描述** |
| ^ | 匹配行首 |
| $ | 匹配行尾 |
| [] | 集合,匹配集合中任意单个字符 |
| [^] | 对集合取反 |
| . | 匹配任意单个字符 |
| \* | 匹配前一个字符任意次数(不允许单独使用) |
| \{n,m\} | 匹配前一个字符n到m次 |
| \{n\} | 匹配前一个字符n次 |
| \{n,\} | 匹配前一个字符n次及以上 |

扩展正则符号列表

|  |  |
| --- | --- |
| **正则符号** | **描述** |
| + | 最少匹配一次 |
| ? | 最多匹配一次 |
| {n,m} | 匹配n到m次 |
| ( ) | 组合为整体,保留 |
| | | 或者 |
| \b | 单词边界 |

## Sed

格式:

前置命令 | sed [选项] ‘条件指令’

sed [选项] ‘条件指令’ 文件...

选项:

-n 屏蔽默认输出,默认sed会输出文档的全部内容

-r 让sed支持扩展正则

-i sed直接修改源文件

1. sed命令的-n选项

执行p打印等过滤操作时，希望看到的是符合条件的文本。但不使用任何选项时，默认会将原始文本一并输出，从而干扰过滤效果。

Sed ‘1p’ /etc/hosts

1. sed命令的-i选项

直接修改源文件

e.g.

打印第3行以及后面的10行

sed -n '3,+10p' /etc/passwd

打印奇数行

sed -n '1~2p' /etc/passwd

指令:

d 删除

删除不包含xml的行

sed '/xml/!d' a.txt

删除文件的最后一行

sed '$d' a.txt

删除所有空行

sed '/^$/d' a.txt

s 替换

s/旧内容/新内容/选项

将每行中的第3个xml替换为XML

sed 's/xml/XML/3' a.txt

将所有的xml都替换为XML

sed 's/xml/XML/g' a.txt

将第4~7行注释掉（行首加#号）

sed '4,7s/^/#/' a.txt

将/bin/bash替换为/sbin/sh

sed 's#/bin/bash#/sbin/sh#' a.txt

每行文本拆分为“第1个字符”、“中间的所有字符”、“倒数第1个字符”三个部分，然后通过替换操作重排顺序为“3-2-1”：

sed -r 's/^(.)(.\*)(.)$/\3\2\1/' nssw.txt

## 3. gawk